A Constraint Logic Programming Approach to Examination Scheduling

Luís Paulo Reis[1,2] and Eugénio Oliveira[1,3]

lpreis@ufp.pt, eco@fe.up.pt

[1]LIACC – Artificial Intelligence and Computer Science Lab. – University of Porto
Http://www.ncc.up.pt/liacc/index.html, Tel: 351-2-2041849, Fax: 351-2-2059280

[2]UFP - CEREM – Multimedia Resource Center, Praça 9 de Abril, 349, 4200 Porto, Portugal

[3]FEUP – DEEC, Rua dos Bragas, 4099 Porto Codex, Portugal

Abstract. The scheduling of exams in institutions of higher education is a large, highly constrained and complex problem. The recent advent of modularity in many universities has resulted in an increase of the complexity of the problem, making the manual achievement of an acceptable solution a tedious and sometimes almost impossible problem. Examination scheduling is the process of assigning exams to time slots in a predetermined period of time and, simultaneously, to assign rooms and invigilators to each exam, satisfying a set of different constraints. This includes avoiding double bookings for rooms, teachers and students, room capacity and type constraints, exam sequence and spreading constraints, preassignments and availability of resources. In this paper we analyse the exam scheduling problem and propose a constraint logic programming approach to solve it. Details concerning problem representation, hard and soft constraints definition and labelling strategies as well as some preliminary results are also discussed.

Keywords: Constraint Logic Programming, Constraint Satisfaction, Scheduling, Examination Timetabling.

1 Introduction

Examination scheduling is the process of assigning exams to time slots in a predetermined period of time and, simultaneously, to assign rooms and invigilators to each exam, satisfying a set of different constraints. This includes avoiding double bookings for rooms, teachers and students, room capacity and type constraints, exam sequence and spreading constraints, preassignments and availability of resources.

The examination scheduling problem is similar to the university course scheduling problem and it is difficult to make a clear and broadly accepted distinction between these two problems. The exact differences depend on the specific problem [13, 17]. However, some major differences are generally accepted:

- There is only one exam to schedule for each course and typically several lectures of different types;
- The schedule period typically greater than a week and the number of periods used may vary;
- Students are always treated individually in exam scheduling;
- In most of the examination problems there can be more than one exam in a room;
- An exam can be split by several rooms;
- Room layout and seat allocation are important in examination scheduling;

- The conflict condition for the students is generally strict: While it is acceptable that a student may have classes overlapped it is not acceptable that a student skips an exam due to exam overlapping;
- Exams for each student must be spread in the timetable while for course timetabling, it is good to group lectures in blocks.

In the last years, one of the major directions for the research on automated (examination) timetabling seems to be the constraint logic programming approach [1, 4, 7, 10]. This conclusion is based on the fact that this technique is able to model very easily any timetabling problem and minor changes in problem formulation are simple to include in the model without significant changes in the program. This, combined with the efficiency of constraint solving methods, enable a system to solve a very large timetabling problem within a small amount of time. Most of the papers concerning the use of CLP for timetabling show only the application to small toy problems or to a single department in a university. However, the recent development of more powerful constraint programming systems [3,12], opens new perspectives for the construction of commercial timetabling systems based on Constraint Logic Programming, capable of tackle complete university timetabling problems.

2 Constraint Logic Programming (CLP) and ECLiPSe

Constraint Logic Programming (CLP) generalises logic programming by replacing unification with constraint solving over a particular domain (e.g. integer, real, finite sets, etc.). This enhancement of logic programming with constraint satisfaction techniques [9], while preserving the declarative nature of logic programming, exhibits surprising efficiency for solving a certain class of combinatorial problems [6] like scheduling, timetabling, resource allocation or circuit verification. A detailed analysis of the application of constraint programming to scheduling can be found in [15]. The usual procedure for solving a Constraint Satisfaction Problem using CLP is to define the set of domain variables Z, together with their domains D, that model all relevant information about the problem. The next step is to specify the constraints C that define the relations between the variables that must hold on the final solution (combinations of values that the variables can take). The last step consists in an enumeration procedure, which together with the internal constraint propagation mechanism and with the backtracking mechanism of the Prolog engine, tries to find an admissible solution for the problem.

The constraint logic programming system ECLiPSe [3] is a successor to the CHIP system [14]. It is a Prolog based system whose aim is to serve as a platform for integrating various Logic Programming extensions, in particular CLP. The Kernel of ECLiPSe is an efficient implementation of standard (Edinburgh like) Prolog [2], build around an incremental compiler, which compiles the source code into WAM-like code [16], and an emulator of this abstract code. Computation in ECLiPSe alternates between two modes: constraint handling and host programming execution (an extended Prolog which handles search and the interaction with the programming environment). The constraint-handling mode extracts all possible information from the constraints, returning execution to the host programming language when no more information can be extracted.

ECLiPSe provides several libraries of constraints solvers which can be used in application programs, such as arithmetic constraint over finite domains, linear rational constraints, Propia (generalised propagation) and Constraint Handling Rules (CHR) that contains itself a library of over 20 further constraints solvers. The system also supports independent development of new constraints solvers form scratch or over existing ones. Eclipse latest release[1] includes the first version of the tkeclipse Development Environment for Unix and Windows. This Environment includes several GUI-based development tools like a program tracer, a term inspector, file and predicate browsers and hypertext help.

3 CSP Model for the Examination Timetabling Problem

In the definition of this model we assume that each module may have one or several teachers responsible and only one exam (given in one or several rooms each with a set of invigilators). We assume that each university degree is composed of a set of obligatory modules. Each teacher has the responsibility of several modules, although each module can have several teachers simultaneous responsible for it. For each teacher his preferences regarding time slots and type of schedule is gathered and recorded in the local database.

There is a set of student types defined. This enables us to model religious, work or any other type of constraints that limit students available time slots. Some religious constraints imply that sets of students are unable to have exams on Saturdays. Some working students can only have exams at night or in the end of the afternoons. For each student type, time preferences are recorded using the same scale used for teacher preferences. The model also includes room types, capacities, availability, localisation and distances.

```
Teacher (#Cod_Teacher, Category, Type, Pref_Schedule).
Student (#Cod_Student, Cod_StudentType).
Student_Types (#Cod_StudentType).
Degree (#Cod_Plan, Cod_Deg).
Module (#Cod_Module, Cod_Plan, Cod_Exam, Year, Semester, Type).
Teacher_Allocation (#Cod_Module, #Cod_Teacher, #Cod_Plan).
Student_Exams (#Cod_Module, #Cod_Student).
Time_Slot (#Cod_Slot, Day, Hour).
Room (#Cod_Room, Cod_RoomType, Capacity).
Room_Type (#Cod_RoomType).
Room_Distance (#Cod_Room1, #Cod_Room2, Distance).
Teacher_Prefs (#Cod_Teacher, #Cod_Slot, Pref).
Student_Prefs (#Cod_StudentType, #Cod_Slot, Pref).
Room_Prefs (#Cod_Room, #Cod_Slot, Pref).
Exams (#Cod_Exam, Num_Students, Cod_RoomType, *Cod_Slot,).
Exam_Room (#Cod_exam, #Number, *Cod_Room).
Exam_Invigilator (#Cod_exam, #Cod_Room, #Number, *Cod_Teacher).
```

Fig. 1. Prolog facts after the generation of the eclipse program.

Each examination is composed of a set of modules. Usually these modules correspond to the same subject (given, for example, to different degrees). However, this kind of modelling enables the definition of exams given at the same time, in the same room (or set of rooms), including several different modules. This is particular useful for very small exams, with only a few students that this way can be grouped into

[1] Eclipse Version 4.1 was released on the 26th of February of 1999

one room, avoiding the waste of room and human resources. An exam takes place in a given time slot, using a set of rooms and a set of invigilators. Each invigilator is assigned for a given room, although several invigilators can supervise the same room.

Each exam is scheduled into one slot variable. For each exam, one or several exam rooms are allocated and for each room, a set of invigilators is defined. This suggests a three phased approach and as a result, we have three different types of variables:

- Time Slot Variables $T=\{T_1, T_2, ..., T_n\}$ where n is the number of examination to be scheduled.
- Room Variables $R=\{R_{11}, R_{12}, ..., R_{1M1}, R_{21}, R_{22}, ..., R_{2M2}, ..., R_{n1}, R_{n2}, ..., R_{nMn}\}$ where n is the number of the examination to be scheduled and $M=\{M_1, M_2, ...M_n\}$ is the maximum number of rooms for each of the n exams.
- Invigilator Variables $I=\{I_{11}, I_{12}, ..., I_{1M1}, I_{21}, I_{22}, ..., I_{2M2}, ..., I_{n1}, I_{n2}, ..., I_{nMn}\}$ where n is the number of exams to be scheduled and $M=\{M_1, M_2, ...M_n\}$ is the number of rooms for each of the exams.

The model considers the following hard constraints:

- C1 - Each exam is scheduled only to one time slot;
- C2 - Each exam has at least one exam room;
- C3 - Each exam room has at least one invigilator;
- C4 - Two exams with a common student must be scheduled in different time slots;
- C5 - Two exams with common teacher/invigilator must be scheduled in different time-slots;
- C6 - A room cannot hold more than one exam at a time;
- C7 - Sum of the capacities of the allocated rooms must be greater than the number of students in an exam;
- C8 - All the rooms allocated to an exam must be of the required type;
- C9 - Each exam has all the responsible teachers as invigilators;
- C10 - A teacher cannot supervise an exam when he/she is unavailable;
- C11 - A student of a given student type cannot have an exam when he is unavailable;
- C11 - A room cannot host an exam when it is unavailable;
- C12 - All prefixed time allocations for exams must me respected;
- C13 - All prefixed room allocations for exams must me respected;
- C14 - All prefixed invigilator allocations for exams must me respected;
- C15 - Some exams must occur in consecutive periods;
- C16 - Some exams have a release date and a due date;
- C17 - Some exams must occur before other exams.

C4 to C6 are general constraints in any scheduling system to avoid resources (students, teachers and rooms) being double booked in a given period. Constraints C7 and C8 are concerned with getting sufficient and adequate room capacity for each exam. Resources unavailability and pre-assignments are treated by constraints C10 to C14, while the last three constrains are concerned with exam ordering.

Usually, a timetabling problem, defined using the hard constraints described above, has no feasible solution! When solving manually a timetabling problem, an experienced planner relaxes some of the constraints, to assure the existence of a solution. This lead to the conclusion that it is very important to include a constraint relaxation system in an automated timetabling system. In a constraint relaxation problem, one must introduce some sort of hierarchy between the constraints [5]. A constraint hierarchy is normally a set of labelled constraints (c@level) where c is a constraint on some variables and level is the strength of c in the hierarchy [11]. In our examination timetabling system, this is achieved by associating a weight to each of the constraints. The higher the value of the weight, the more important the constraint is and, thus, less adequate to relaxation. Constraints with weight 100, correspond to mandatory (hard) constraints and cannot be relaxed. Our constraint relaxation scheme is at the moment very simple and is based only in treating all the constraints (except those with weight 100) as soft constraints. The weights

are used, in conjunction with the evaluation criteria to construct a global function to be minimised. This way, for the hard constraints we have the function:

$$Evaluation_1(P) = \sum_{i=4}^{17} (W_i nC_i) \quad (1)$$

in which nC_i corresponds to the number of constraints of type C_i that are not respected and W_i corresponds to the weight of constraint i. The soft constraints used as evaluation criteria are:

- S1 - Number of periods between two examinations for a student (tabled);
- S2 - Number of examinations that a student has in a given day (tabled);
- S3 - Number of examinations that a student has in two consecutive days (tabled);
- S4 - Number of teacher preference disrespected (weighted);
- S5 - Number of student preference disrespected (weighted);
- S6 - Number of rooms' preference disrespected (weighted);
- S7 - Total number of rooms x periods used;
- S8 - Total number of invigilators x periods used;
- S9 - Distance between Rooms of the same examination;
- S10 - Respect for the teachers desired schedule (concentrated/disperse);
- S11 - Number of extra exams (from the desired) that each teacher has to supervise.

Constraints S1, S2 and S3 have tabled values defined by the user. Great penalty values are assigned for small time intervals between student consecutive examinations and for large number of exams in a given day (table 2) or in consecutive days.

Table 1. Number of examinations in a day for students (constraint S2).

Examinations on a Given Day	0	1	2	3	4	5
Penalty	$S2_1$	$S2_2$	$S2_3$	$S2_4$	$S2_5$	$S2_6$
Default Values	0	0	2	4	6	8

Formula 2 is used to weight the evaluation criteria:

$$Evaluation_2(P) = \sum_{i=1}^{11} (W_i f(S_i)) \quad (2)$$

in which $f_i(S_i)$ corresponds to a function of the value achieved for S_i and W_i corresponds to the weight associated with constraint i. Combining functions (1) and (2) we have the global evaluation function to be minimised:

$$Evaluation(P) = W_H \times Evaluation_1(P) + W_S \times Evaluation_2(P) \quad (3)$$

Four main labelling strategies are included in the system. The first strategy begins by dealing with time slot variables and then proceeds to the instantiation of room variables and, at the end, deals with invigilator variables. The second strategy, begins with room variables, then time slot variables and at the end invigilators. The third strategy, after finishing time slot variable labelling, proceeds to simultaneous room and invigilator variable labelling. In the last general strategy, the three types of variables are all dealt simultaneously. For dealing with the optimisation problem (using the evaluation function described), we use standard Branch and Bound algorithm. For variable ordering we use a dynamic ordering method using the first-fail heuristic [8]. Specific heuristics are included for room and invigilator variables ordering. For value ordering, we use random value selection with appropriate spreading for time slot variables and a heuristic based on inverse capacity order for room values.

4 Experimental Results

The examination timetabling system developed was applied to the data available regarding the previous semester in University Fernando Pessoa in Porto. This university has a total of more than 3500 students, 230 teachers (115 of which had to supervise exams), 21 different degrees and almost 500 different modules corresponding to 314 examinations. More than two thousand students had six or more examinations to perform and more than three hundred had nine or more exams. The examination period used corresponds to three weeks, each with four possible time slots corresponding to a total of 66 time slots. 33 rooms (most of them with small capacity) could be used for the exams. Our preliminary results for the time slot allocation examination problem has shown that the problem analysed could be solved without violating the hard constraints imposed. Using the manual resolution method, this result was also possible but it needed almost a week of planning efforts.

In our experiments, all the hard constraints were used as typical hard constraints (meaning that any feasible solution must respect them) and we used the following evaluation function for the solution:

$$Evaluation(P) = f(S2) + 5 * S4 + 5 * S5 + 5 * S6 + S7 + 10 * S11 \qquad (4)$$

Large weights were used for S4, S5, S6 and S11 since those soft constraints are easy to respect in our problem. Constraints S2 and S7 were used as our main evaluation of the final solution. The final values achieved (after 20 minutes of computing in a Pentium III 450MHz) are depicted in table 2.

Table 2. Evaluation of the final solutions achieved after 20 minutes of computing.

Constraint	S2	S4	S5	S6	S7	S11	Total
Strategy1 (T→R→I)	32	3	0	6	526	0	603
Strategy2 (R→T→I)	86	16	0	0	524	0	690
Strategy3 (T→R,I)	32	3	0	6	526	0	603
Strategy4 (T,R,I)	102	12	0	6	540	0	732
Manual Solution	1314	48	0	16	564	0	2198

The solutions achieved by our method have global evaluations more than three times better than the manual solution obtained. Major differences are located in soft constraints S2 and S4. By solving manually the problem it is almost impossible to respect soft constraint S2 due to the enormous amount of cross information involved. For constraint S4, the difference between the evaluations of the manual and automatic solutions is due to the fact that this constraint was somewhat overlooked by the problem solvers when solving the problem manually.

Labeling strategies 1 and 3 find the same solution after 20 minutes (although strategy 1 is slightly faster). Strategies 2 and 4 are not able to find very good solutions (following the proposed evaluation criteria). This is due to some non optimal allocations that are performed in the beginning of the labeling process concerning rooms (in strategy 2) and invigilators (in both strategies). Those allocations are not changed due to the lack of time to perform a deep backtrack. In other experiments, performed using small random generated problems, we conclude that for typical examination timetabling problems, the best strategies are 1 and 3. Strategy 2 is very good for problems with very constrained space allocation but

performs poorly in problems with a high percentage of conflicting exams between students. Strategy 4 is only good for problems concerning non modular universities (in which groups of students have similar degree programs) with tight constraints for rooms and invigilators. Other strategies, in which invigilator variables are instantiated first, do not perform well in any kind of examination timetabling problems.

5 Conclusions and Future Work

In this paper we have presented the examination timetabling problem, its differences when compared with the general timetabling problem and a new constraint satisfaction model for the problem. The constraint satisfaction model proposed works on a very simple but yet powerful problem information model and enables split examinations, distance analysis between examination room and invigilator assignment in a three phased approach. The preliminary results achieved for a large, real examination timetabling problem show the usefulness and scalability of the proposed model. Future work will be concerned with the development of new labelling strategies, integration of constraint relaxation mechanisms, analysis of new real problems and standard benchmarks and extension of the system using constraints over sets and repair heuristics.

Acknowledgement

This work was partially supported by a research grant PRAXIS XXI, BD/5663/95 of the Portuguese Foundation for Science and Technology.

References

1. Azevedo F. and Barahona P., Timetabling in Constraint Logic Programming, Proceedings of World Congress on Expert Systems'94, January of
2. Clocksin W., and Mellish C., Programming in Prolog, Springer-Verlag, 1981
3. Eclipse User Manual, Aggoun et al., ECRC GmbH, 1992, Int.l Computers Limited and IC-Parc, 1998
4. Frangouli H. ; Harmandas V. and Stamatopoulos P., UTSE: Construction of Optimum Timetables for University Courses - A CLP based Approach, Third International Conference on the Practical Applications of Prolog, PAP'95, Paris, pp.225-243, April 1995
5. Freuder E., Partial Constraint Satisfaction, Proceeding of the 11th International Joint Conference on Artificial Intelligence, IJCAI89, pp. 278-283, Detroit, 1989
6. Fruhwirth, T. et al, Constraint Logic Programming – An Informal Introduction, Lecture Notes in Computer Science, Vol. 636, Logic Programming in Action, pp. 3-35, 1992
7. Gueret, Christelle and Jussien, Narendra and Boizumault, Patrice and Prins, Christian, Building University Timetables Using Constraint Logic Programming, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (PATAT '95), pp.393-408, 1995
8. Haralick, R. and Elliott, G., Increasing Tree Search Efficiency for Constraint Satisfaction Problems, Artificial Intelligence, Vol. 14, pp. 263-313, 1980
9. Hentenryck, P. Van, Constraint Satisfaction in Logic Programming, Logic Programming Series, MIT Press, Cambridge, MA, 1989
10. Lajos, Gyuri, Complete University Modular Timetabling Using Constraint Logic Programming, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling PATAT'95, Edinburgh, pp. 364-375, 1995
11. Menezes, F. ; Barahona, P. and Codognet, P., An Incremental Hierarchical Constraint Solver Applied to a Time-Tabling Problem, Proceedings of the Thirteenth International Conference on Artificial Intelligence, Expert Systems and Natural language, Avignon, France, Vol. 1, pp. 297-306, May 1993
12. DFKI Oz User's Manual, Mehl M. et al. DFKI, Programming System Lab., German Research Center for Artificial Intelligence, 1998
13. Schaerf, A., A Survey of Automated Timetabling, Technical Report CS-R9567, CWI – Centrum voor Wiskunde en Informatica, 1995
14. Simonis H., The CHIP System and its Applications, Montonari and Rossi, editors, Proc. Principles and Practice of Constraints Programming, pp.643-646, Springer-Verlag, 1995
15. Wallace, M. Applying Constraints for Scheduling, In B.Mayoh and J. Penjaam, editors, Constraint Programming: Proceedings 1993 NATO ASI Parnu, Estonia, pp. 161-180, Springer-Verlag, 1994
16. Warren D., An Abstract Prolog Instruction Set, Technical Note 309, SRI, October, 1983
17. Weare, R., Automated Examination Timetabling, PhD Thesis, Department of Computer Science, University Nottingham, U.K., June 1995